Universitat de Lleida

TREBALL FINAL DE GRAU

ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Albert Font Farré

Titulació: Grau en Enginyeria Electrònica Industrial i Automàtica

Títol de Treball Final de Grau: **Disseny d'una estació meteorològica remota i un sistema de monitorització**

Director/a: **Tomàs Pallejà Cabré**

Presentació

Mes: Juliol

Any: 2021

Índex

1 Introducció

La meteorologia és la ciència interdisciplinària de la física de l'atmosfera, que estudia l'estat del temps, el medi atmosfèric, els fenòmens que s'hi produeixen i les lleis que el regeixen. És un àmbit de molta importància saber en tot moment quines condicions climàtiques hi ha en certes zones, com per exemple, camps de conreu, per a obtenir-ne el màxim benefici i productivitat.

Aquest projecte pretén ajudar a conèixer les principals magnituds meteorològiques, facilitant informació actualitzada sobre la temperatura, humitat de l'aire, pressió atmosfèrica, direcció i velocitat del vent i precipitació acumulada. El seguiment d'aquestes variables climàtiques es farà utilitzant sensors, els quals transformen les magnituds físiques a un senyal elèctrica. Aquest fet fa que una part important del projecte sigui escollir apropiadament els sensors i condicionar els senyals que n'obtenim per a interpretar-les i fer un seguiment correcte.

Les dades obtingudes seran interpretades per un microcontrolador, en l'actual projecte, l'Arduino Mega 2560. El microcontrolador rep totes les dades que recullen els sensors i les envia a un servidor extern, a través de la telefonia mòbil, mitjançant el protocol de comunicació HTTP i des d'un mòdul, estil shield (escut) de la marca DFRobots amb un perifèric SIM7000E, en el qual es pot posar una targeta SIM de mòbil normal i corrent (per al disseny i funcionament he fet servir la meva pròpia targeta).

El servidor extern, junt amb una base de dades, permetrà l'emmagatzematge i posterior consulta de totes les magnituds. La creació del servidor permet un accés a la informació recollida per l'Arduino en tot moment, a més d'un històric per a poder-les consultar quan es vulgui.

Per a finalitzar la introducció, a continuació es proposa un diagrama amb les parts del projecte, que servirà per a completar l'explicació:

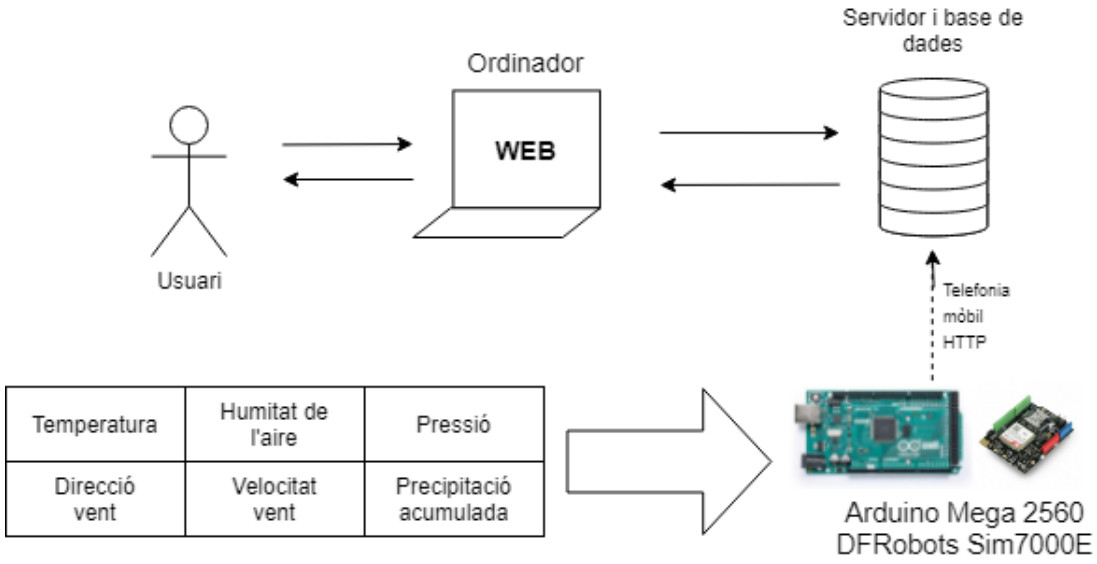


Fig. 1. Esquema funcionament

2 Objectius

L'objectiu d'aquest projecte és dissenyar i construir una estació meteorològica remota i un servidor amb base de dades per poder consultar la informació obtinguda. Es vol fer el disseny de totes les parts des de zero, sense fer ús de tecnologies més avançades, per tal de saber i completar tot el procediment de la creació d'un prototip.

Aquest dispositiu està enfocat a un ús en camps de conreu, per a monitorar l'estat meteorològic que es troba el camp on se situï l'estació meteorològica. Per aquesta raó, també es vol dissenyar un servidor amb una base de dades, on es podrà consultar l'estat actual del camp i tot el registre històric de dades.

Pel ús que ha de tenir i la localització en què s'haurà de situar l'estació meteorològica, s'ha de fer un disseny que sigui capaç de suportar les condicions adverses sota les quals es pugui trobar, sigui calor abrasiu o fred intens, i, fins i tot, pluja o neu. És molt important conèixer el consum del dispositiu durant el seu funcionament, reduint al màxim el consum de cada component quan no sigui necessari el seu ús i poder elegir una font d'alimentació correcta.

Així doncs, es tracta d'un projecte interdisciplinari, en el qual s'ha d'integrar la comunicació entre el perifèric SIM7000E i un servidor, la interpretació de les dades obtingudes pels sensors i el seu posterior anàlisi i la visualització de tota la informació emmagatzemada de forma que qualsevol usuari ho pugui visualitzar. Per a fer-ho s'hauran d'aplicar coneixements d'electrònica que s'han anat treballant durant el grau, i adquirir-ne de nous d'altres disciplines com la informàtica o les telecomunicacions. S'ha de tenir en compte tot el que comporta treballar amb tecnologies que no hi estic familiaritzat i com pot afectar el temps invertit a adquirir nous coneixements al dispositiu que se n'obtindrà.

3 Materials i mètodes

3.1 Introducció

Les estacions meteorològiques estan formades per diferents sensors de precisió. Depèn de les dades que es vol obtenir, s'han d'adquirir uns sensors o uns altres. Fent una petita cerca, es pot observar una primera classificació: estacions meteorològiques domèstiques i professionals.

Les estacions meteorològiques domèstiques, normalment ens proporcionen informació sobre la temperatura interior i exterior, la pressió i la humitat relativa. La seva estructura es força simple i en la majoria només s'hi mostren els valors actuals i el màxim i mínim diaris. Normalment estan situades a l'interior de les cases, amb un sensor de temperatura situat a l'exterior, que es comunica de sense fils amb el mòdul principal, que es troba a l'interior de les cases, des d'on es mostren les dades. A vegades mostren prediccions, que obtenen a través d'internet, connectades a algun servei de meteorologia.

Pel que fa a les estacions professionals, es caracteritzen per tenir un nombre més elevat de sensors, així com un sistema d'emmagatzematge per a poder fer un seguiment i moltes vegades les consultes es poden fer de forma remota. A part, les condicions que han de suportar són molt més adverses, ja que tota l'estació meteorològica és un sol dispositiu i es col·loca a l'exterior. El seu ús està dedicat a l'obtenció de forma rigorosa de les condicions climàtiques per ala seva anàlisi.

Amb tot això, es veu clarament que l'estació meteorològica que es vol dissenyar forma part de les estacions meteorològiques professionals.

3.2 Maquinari

Per al disseny del projecte que es vol dur a terme, és necessari conèixer les característiques dels components electrònics que es faran servir, per tal que no ens donin problemes a llarg termini.

Per tal d'analitzar els valors que obtenim de cada sensor, és necessària la utilització d'un microcontrolador que rebi els senyals, analògiques i digitals i les transformi en valors de magnituds que puguin ser interpretades per l'usuari.

Amb les dades recollides, aquestes s'han de poder enviar de forma remota al servidor extern, el qual es pot trobar situat a llargues distàncies de la mateixa estació. Aquesta funció la desenvoluparà el mòdul DFRobot que porta incorporat el perifèric SIM7000E, on es poden posar targetes SIM de mòbils per tal de fer els enviaments.

3.2.1 Arduino MEGA 2560

El microcontrolador que s'ha elegit perquè transformi els senyals dels sensors en dades interpretables és l'Arduino MEGA 2560.

Aquest microcontrolador està basat en el processador ATmeg2560. Posseeix un total de 54 entrades i sortides digitals, dels quals 15 es poden usar com a senyals de sorida PWM (Pulse-Width Modulation); també incorpora 16 entrades

analògiques, 4 ports USART (connexions tipus hardware mitjançant port sèrie), un oscil·lador de 16 MHz, connexió USB, un connector de potència tipus Jack, un capçal ICSP (In Chip Serial Progamer), etc.

Una de les funcionalitats que més ens interessa és que és compatible amb molts dels mòduls shields dissenyats per a l'Arduino UNO. Aquesta peculiaritat ens és de gran utilitat, ja que el mòdul de DFRobots del que disposem està pensat per a fer-se anar en l'Arduino UNO.

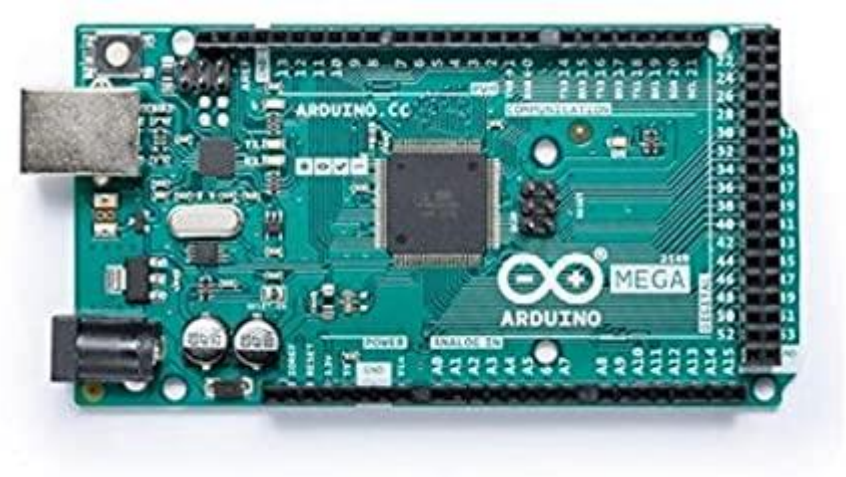


Fig. 2. Arduino MEGA

3.2.2 DFRobot_SIM7000E

Per a poder transmetre les dades de l'estació meteorològica al servidor i emmagatzemar-les a la base de dades per la seva posterior consulta, es fa servir el mòdul DFRobots SIM 7000E, una shield dissenyada per a l'Arduino UNO i compatible amb l'Arduino MEGA i desenvolupada per la marca DFRobots.

Aquest mòdul de comunicació sense fil suporta comunicació de multifreqüència LTE-FDD, freqüència dual GPRS/EDGE i comunicació NB-IoT. Ha estat pensat per a qualsevol classe de prototip de baix consum. El xip SIM7000E és compatible tant amb targetes SIM tradicionals com amb targetes de NB-IoT.

Això ens permet fer aplicacions típiques de GSM, com enviar SMS o rebre una trucada, però també podem fer transmissions del tipus NB-IoT, com les del protocol de MQTT.

A part del perifèric SIM 7000E també conté un sensor de posicionament global que suporta GPS, GLONASS, Galileo, QZSS i BeiDou; i un sensor ambiental BME280 que pot monitorar la temperatura, humitat i pressió.

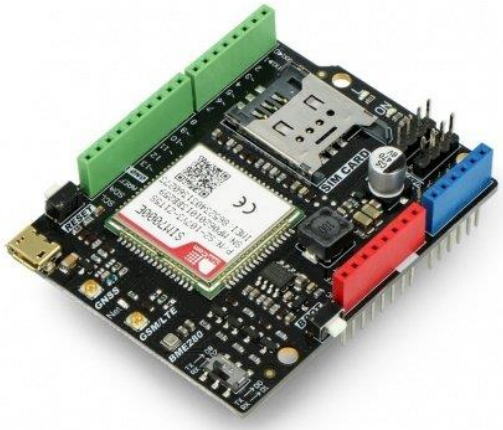


Fig. 3. Mòdul Shield de DFRobots amb SIM_7000E i el sensor BME 280

3.3 Sensors

3.3.1 Temperatura

La temperatura és una magnitud física que determina el grau d'excitació de les partícules, es troba relacionada directament amb l'energia cinètica de les partícules i molècules que conformen la matèria que es vol mesurar.

En termes meteorològics ens interessa conèixer la temperatura seca, que és aquella que no té en consideració els efectes de la radiació calorífica que desprenen els objectes que envolten l'ambient i dels efectes de la humitat relativa i de la velocitat de l'aire.

Des d'un punt de vista agrícola, la temperatura és indubtablement important per a conèixer l'estat dels camps i evitar d'anys majors.

Per a mesurar aquest valor es farà servir el sensor de temperatura incorporat en el perifèric BME de la shield de DFRobots. A l'estar incorporat amb el mòdul, l'obtenció de la temperatura se simplifica molt. En els codis d'exemple de la pàgina web del fabricant hi ha un codi per a aconseguir el valor de temperatura que llegeix el sensor.

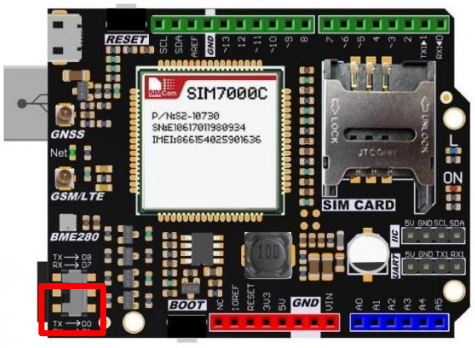


Fig. 4. Senyalització sensor ambiental BME 280

3.3.2 Humitat relativa atmosfèrica

L'aigua és un element abundant en el planeta terra, la qual es troba en constant rotació seguint l'anomenat cicle de l'aigua: l'aigua s'evapora, es condensa en l'atmosfera i es precipita en forma de pluja, neu o calamarsa per a tornar a començar. Quan l'aigua es condensa es formen núvols, boira o rosada i és la causa de què hi hagi humitat en l'aire.

Hi ha tres magnituds que ens indiques el vapor d'aigua que hi ha en un cert volum d'aire: la humitat absoluta, l'específica i la relativa. En el nostre projecte, la magnitud que ens interessa és la humitat relativa atmosfèrica, la qual es defineix com el quocient amb tant per cent entre la humitat absoluta (massa de vapor d'aigua per volum) de l'aire a una certa temperatura i la que el mateix aire tindria si estigués saturat a la temperatura que es trobi, és a dir, la màxima humitat que pogués contenir.

En l'agricultura, la humitat relativa atmosfèrica, és útil per a conèixer en quin estat es troben les fulles i tiges dels arbres i plantes. Sobretot és important tenir un històric de la humitat atmosfèrica per a saber amb precisió la quantitat d'aigua que serà necessària per a afavorir el creixement de la collita

El sensor d'humitat que es fa servir també es troba incorporat en el perifèric BME i se n'obté la informació fent servir el mateix codi d'exemple que en el cas anterior.

3.3.3 Pressió atmosfèrica

La pressió atmosfèrica és la força per unitat de superfície que exerceix l'aire de l'atmosfera sobre la superfície de la Terra. La variació de la pressió atmosfèrica pot ser deguda a un canvi d'altura (si ens movem) o venir associada a un canvi meteorològic, d'aquí la importància de monitorar els canvis de pressió, ja que és un indicador d'un canvi de temps o d'un període d'inestabilitat climàtica.

Per a mesurar la pressió atmosfèrica es fa servir el sensor que incorpora el mòdul shield de DFRobots, situat en el perifèric BME, d'on obtenim el valor a través del codi d'exemple de la pàgina web del fabricant.

3.3.4 Velocitat i direcció del vent

El vent és el moviment de masses d'aire de zones d'altes pressions (anticiclons) a zones de baixes pressions (depressions o borrasques), per tant, la direcció que té el vent acostuma a ser d'una zona d'altres pressions a una de baixes, i en zones sense canvis de pressió el vent tendeix a desaparèixer.

Per a mesurar la velocitat del vent es fa servir el que s'anomena anemòmetre, un instrument que gira depenent de la velocitat del vent, com més vent fa, més ràpid gira, i si no en fa, no gira. La direcció del vent es mesura amb un penell, un penell simplement assenyala la direcció del vent, generalment s'expressa amb una rosa dels vents.

Fig. 5. Anemòmetre

Fig. 6. Penell

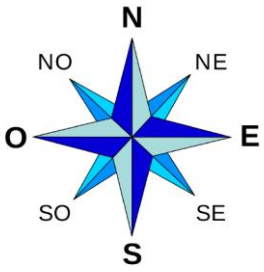


Fig. 7. Rosa dels vents

3.3.5 Pluja acumulada

Quan l'aigua es condensa per a formar núvols, aquests comencen a ascendir i a refredar-se, a mesura que es van refredant, la mida de les gotes va augmentant fins que el seu elevat pes les fa precipitar cap a la superfície terrestre, donant lloc al qual anomenem pluja. La pluja es mesura amb litres caiguts per metre quadrat de superfície.

Per a mesurar la pluja es fa anar un pluviòmetre, un instrument que recull l'aigua que cau en una certa superfície i estima la que ha caigut a la zona si la intensitat de pluja és uniforme a la que s'ha precipitat a sobre de l'instrument.

Fig. 8. Pluviòmetre

Per a mesurar la velocitat i direcció del vent i la pluja acumulada, es fa servir una estació meteorològica de la marca SparkFun, amb un anemòmetre, un penell i un pluviòmetre, els quals es poden programar de forma molt senzilla amb l'Arduino i que tenen molt bona fiabilitat.

Fig. 9. Estació meteorològica SparkFun

3.3.6 Obtenció de dades

3.3.6.1 **BME 280**

Una vegada es coneixen els sensors que s'han de fer servir, s'han d'obtenir les dades. Els sensors que trobem integrats en el mateix mòdul de DFRobots (temperatura, pressió i humitat) obtenim els valors directament del codi d'exemple que trobem a la llibreria del fabricant.

En el codi d'exemple també dona informació sobre l'altitud a què es troba el dispositiu, per a fer-ho es basa en la pressió a nivell de mar, que és una constant que se li ha d'indicar a l'inici del codi, en relació amb la pressió que es mesura.

3.3.6.2 **Anemòmetre**

Per a mesurar la velocitat del vent es fa servir l'anemòmetre de l'estació meteorològica SparkFun, en el qual es fa servir un sensor electromagnètic, per tant analògic, tot i que, tenint en compte el funcionament del sensor, no és necessària una conversió analògica-digital. Això es deu al fet que l'anemòmetre mesura la velocitat del vent a través d'un contacte magnètic el qual provoca que s'obri i es tanqui un interruptor. Cada vegada que es provoca un cicle de tancament en un segon el vent té una velocitat de 2,4 km/h. Per tant aquest sensor provocarà trens de polsos quan hi hagi vent.

Per al muntatge simplement fa falta una resistència pull-up o pull-down (és indiferent quina, ja que hem de mesurar polsos complets, no pujades o baixades), en el cas de l'anemòmetre, serà pull-down i es dissenya manualment. Això es pot connectar directament a una entrada digital, en la que s'haurà de mirar si es troba a nivell alt o baix, i cada vegada que això succeeixi, haurà passat un pols. Per a tenir una mesura més fiable, es conten els polsos que hi ha durant 5 segons, es divideix entre el temps que s'han estat mesurant i es multiplica per 2,4.

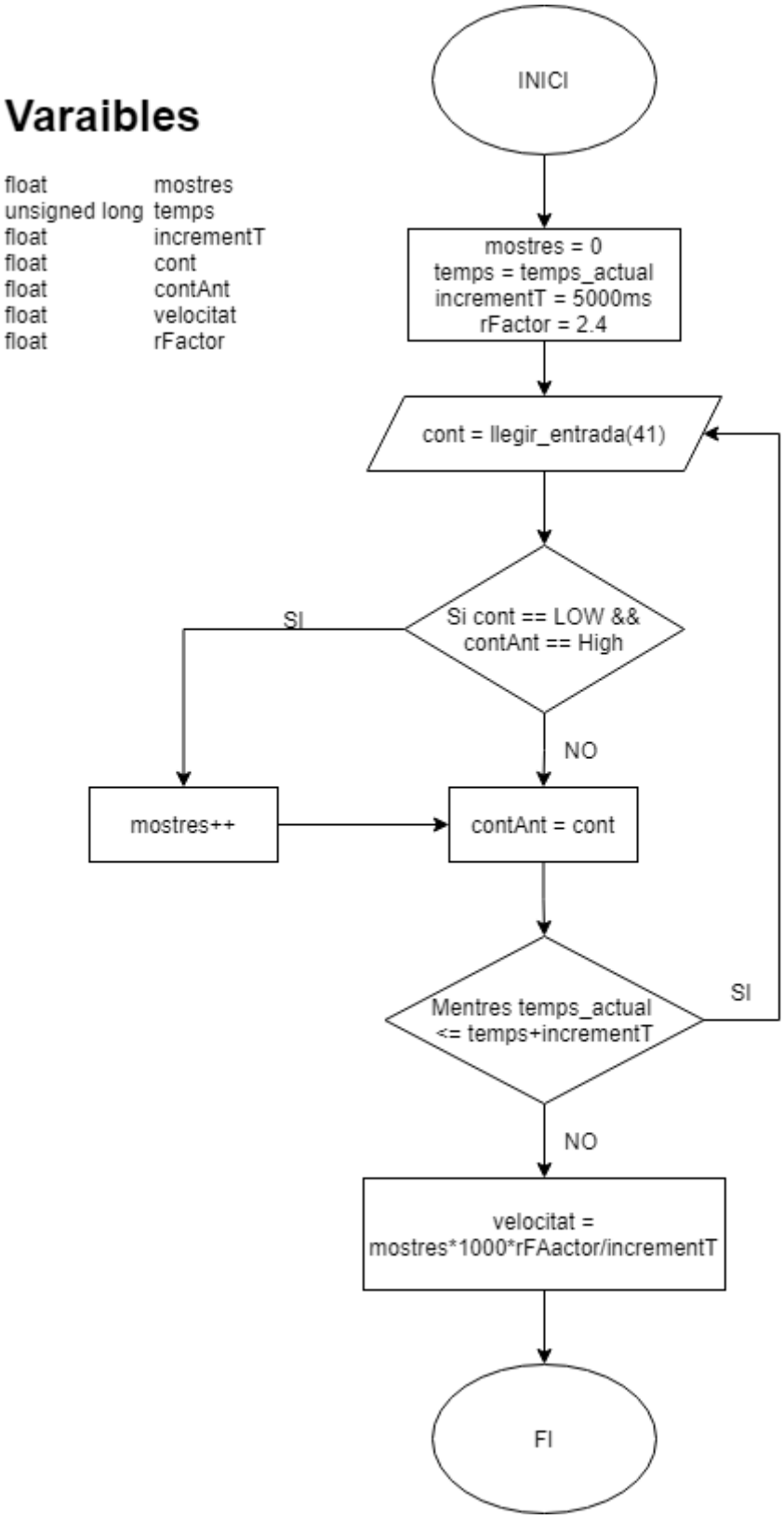


Fig 10. Diagrama de Flux funcionament anemòmetre

3.3.6.3 **Penell**

El penell ens indicarà la direcció del vent. El penell del qual disposem es tracta d'un sensor electromagnètic que varia la resistència en modificar-se a la direcció del vent, tenint vuit valors de resistència diferents, per a vuit direccions del vent (N- S- E- W- NE- NW- SE- SW). En aquest cas tenim un sensor analògic, per tant s'haurà de fer servir una entrada analògica, la qual ens indicarà, respecte a cinc volts, quin voltatge de sortida hi ha en els borns de la resistència variable del penell, en una escala que va des de zero fins a la potència desena de dos.

El fabricant ens dóna informació sobre el voltatge en els borns de la resistència que s'obté per cada posició amb una tensió de treball de 5 volts, sense indicar quin ha de ser el valor de la resistència fixa, per tant, es van fer proves per a mesurar quin valor d'entrada entre 0 i 1024 (potencia desena de dos), obtenim per cada posició del penell així assegurar que sempre obtindrem la informació correcta.

D'aquesta manera, el disseny per a obtenir el valor de la direcció del vent es basarà en un divisor de tensió on hi haurà una resistència fixa coneguda i el valor de la resistència interna del penell, que variarà en funció d'una de les vuit direccions on apunti. Mesurant la caiguda de tensió en qualsevol dels dos elements i apuntant el valor que obtenim a l'Arduino, ja es pot crear una funció que en doni la direcció del vent.

En el codi es dóna un petit rang per a cada posició, per tal d'evitar problemes amb el soroll o variacions del valor per culpa de la temperatura.

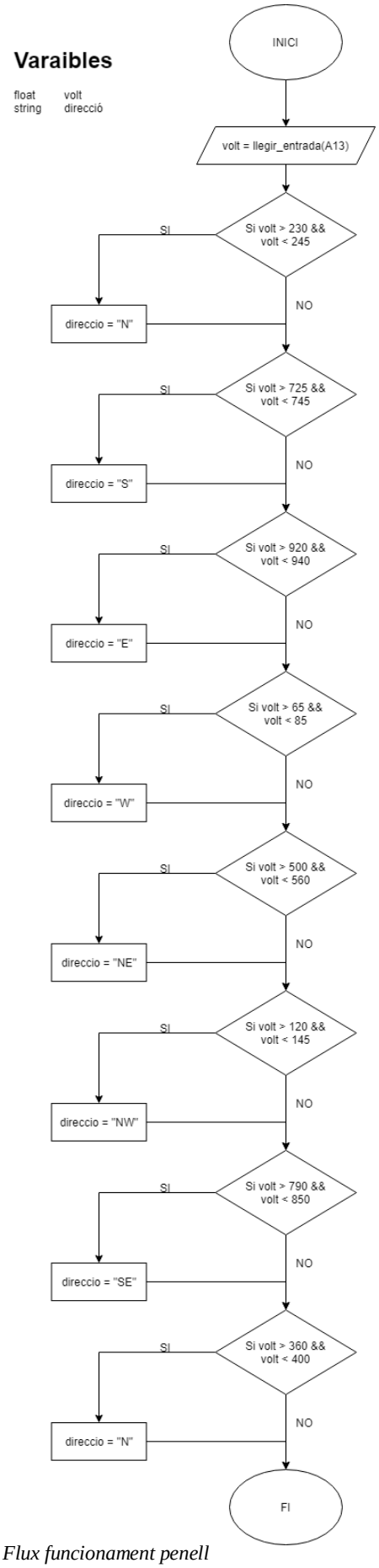


Fig. 11. Diagrama de Flux funcionament penell

3.3.6.4 **Pluviòmetre**

Per a saber la pluja acumulada es farà servir el pluviòmetre electromecànic, amb un funcionament molt similar al de l'anemòmetre ja comentat, ja que cada vegada que recull una certa quantitat d'aigua, s'obra i es tanca un contacte magnètic, per tant el seu comportament és molt similar a l'un interruptor. A causa del seu comportament, es tracta d'un sensor analògic que no necessita un convertidor analògic-digital, ja que el que rebem del sensor és un pols cada vegada que el sensor rebi 0,2794 mm de precipitació.

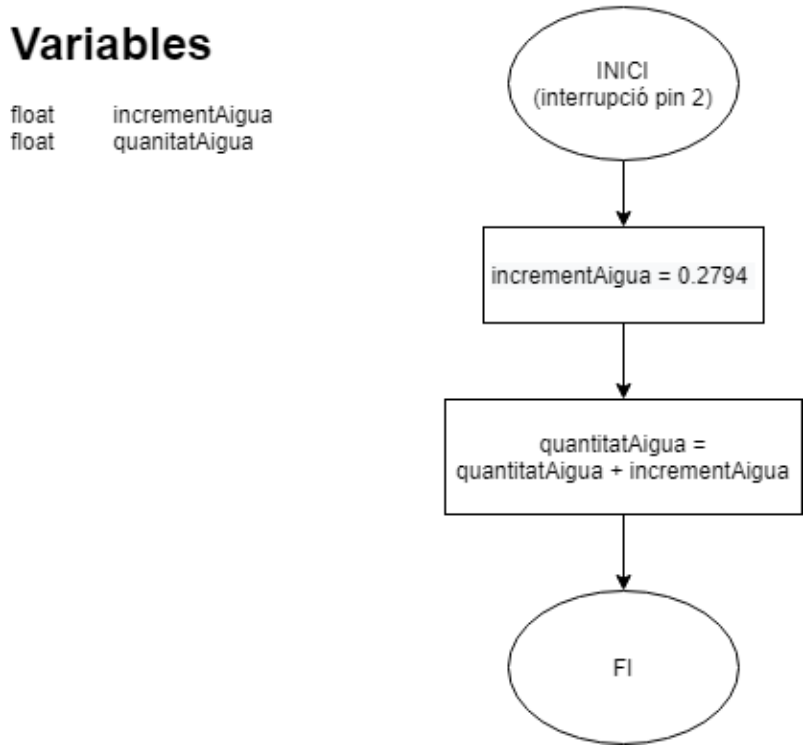


Fig. 12. Diagrama de Flux funcionament del pluviòmetre

3.4 Programari

Perquè el projecte tiri endavant, és molt important tenir una bona comunicació entre el servidor i la base de dades, i l'estació meteorològica (concretament amb el mòdul de DFRobots, que es programa a través de l'Arduino).

3.4.1 Servidor

Un servidor d'un punt de vista informàtic és un ordinador, o un conjunt d'ordinadors que són capaços d'atendre les peticions d'un client i retornar una resposta en concordança.

En aquest treball, el servidor és la peça més important de tot el programari: és el que rep les dades meteorològiques de l'estació, les emmagatzema i posteriorment és des d'on són accessibles pel client. Tant la comunicació Arduino- Servidor com la visualització de les dades Client- Servidor es duen a terme a través d'enllaços del mateix servidor.

3.4.1.1 **Django**

Django és un tipus d'estructura web (web framework) de codi obert i basat en el llenguatge Pyhton, que segueix un estil model- plantilla- vistes (MTV de l'anglès Model- Template- Views).

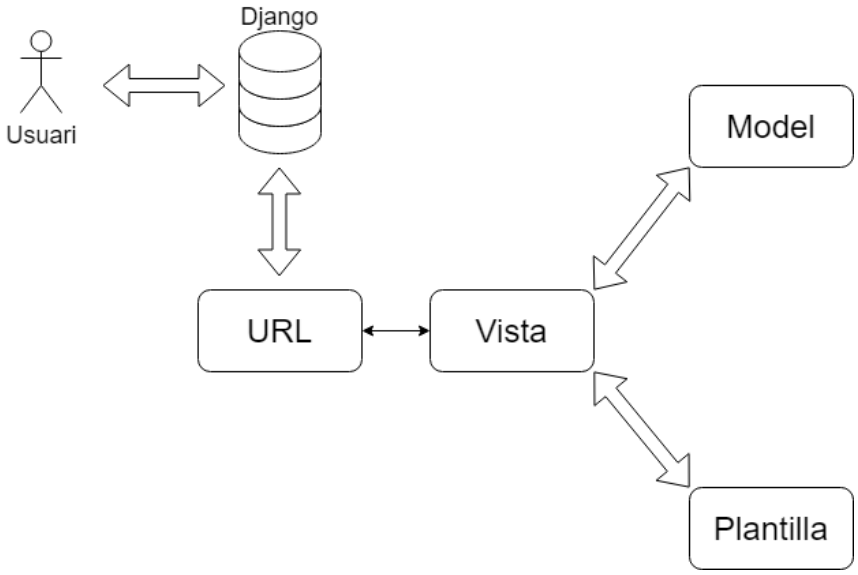


Fig. 13. Estructura web basada en un format MTV

Una estructura web és un sol programa que facilita la creació de pàgines webs a través d'una integració completa entre servidor i base de dades. Per aquest motiu el principal objectiu de Django és facilitar la creació de pàgines webs complexes que funcionin amb una base de dades guiada.

El llenguatge Python es fa servir per a tota la programació del servidor: configuració, documents i models de dades.

Django permet, amb un mateix llenguatge i de forma molt organitzada: la creació d'un model de dades (arxiu models.py), indicar la funció de cada pàgina o vista a través de classes o mètodes pròpies de Python (arxiu views.py), unificació de les adreces que tenen cada vista (arxiu urls.py) i les visualitzacions de les pàgines amb llenguatge HTML, CSS i altres llenguatges que representin sistemàtiques de representació de documents.

Altres coses que es poden fer és la creació d'aplicacions. Cada aplicació tindrà el seu propi estil MTV i els arxius prèviament explicats. És una forma molt efectiva que te Django per tal de poder treballar conjuntament programadors i dissenyadors en espais diferents però sobre el mateix servidor. Totes les aplicacions que es creen amb Django són controlades des de la carpeta principal que du el mateix nom que el projecte.

Molts llocs de la xarxa fan servir Django, com ara Instagram, Mozilla, The Washington Times, Bitbucket i més.

Fig. 14. Logotip estructura web Django

Per a la creació del servidor s'ha fet servir la versió 3.2.2 de Django i la 3.12.4 de l'estructura web djangorestframework.

3.4.1.2 Base de dades: SQLite

Abans d'entrar a explicar la base de dades que s'ha utilitzat, es farà una petita introducció al que són les dades de dades:

Una base de dades és un conjunt de dades que segueixen una estructura coherent i accessibles des d'un o més programes o aplicacions, de manera que qualsevol d'aquestes dades pot ser extreta del conjunt i actualitzada, sense afectar l'estructura del conjunt ni les altres dades.

Una base de dades pot seguir un model de dades. Un dels més comuns és el model relacional de dades i el llenguatge més conegut per a aquests models relacionals és el SQL. Una base de dades que segueixi un model relacional converteix l'ordinador en un gran administrador d'arxius de dades, cosa que ens permet extreure diferents tipus d'informació de manera ràpida i fàcil. Les dades s'emmagatzemen en taules, les quals tenen diferents camps o columnes que són les dades que s'emmagatzemaran.

De la base de dades SQLite és important anomenar que és un sistema de gestió de bases de dades relacional de fins a 2 Terabytes de mida.

Fig. 15. Logotip base de dades SQLite

3.4.1.3 **Procediment creació Servidor- Base de dades**

El funcionament i la posada en marxa del servidor es va dur a terme amb l'ajut imprescindible del Guillem Felis, company que està cursant 4rt del grau d'enginyeria informàtica. Sense ell jo no hauria sabut com dissenyar el servidor, donat que jo no havia fet servir mai aquestes tecnologies.

Els primers passos van ser la instal·lació d'una màquina virtual amb sistema operatiu Ubuntu 20.04.2 LTS, en aquesta màquina s'hi va instal·lar un entorn virtual de treball amb l'eina Pipenv. D'aquesta manera, qualsevol error que es pogués produir no repercutiria a la màquina virtual (s'ha de notar que, al ja estar treballant en una màquina virtual, si s'hagués produït cap error greu en el projecte i aquest hagués danyat la màquina virtual, l'ordinador no s'hagués vist compromès, ja que el problema hagués seguit encapsulat en la màquina virtual).

Amb l'entorn de treball creat i un cop en ell, es va procedir a la instal·lació del Django. Per a entendre el funcionament del programa, es van seguir els llibres recomanats pel Guillem: *Django for Begginers*, *Django for Professionals* i *Django for APIs Build Web APIs with Python*, en format PDF de l'autor *William S. Vincent*. Gràcies a aquests llibres i a pautes que el Guillem m'anava donant, es va poder entendre el funcionament bàsic de les estructures web amb un estil MTV.

A continuació, i després d'uns quants projectes, es va dur a terme el projecte definitiu, amb una sola aplicació, on es va crear la base de dades que finalment s'ha fet servir i els diferents enllaços per a obtenir la visualització de les dades o des d'on poder-les actualitzar.

Pel que fa a la base de dades, la creació amb Django és molt simple i ràpida: es defineix un model (una classe de Python) i s'apliquen les migracions. Aplicar migracions de les aplicacions és com l'estructura web Django anomena als processos de creació que calen dur-se a terme per a crear i actualitzar les bases de dades.

Per al meu projecte, he creat dues classes, és a dir, dues taules. La primera serveix per a emmagatzemar cada estació meteorològica que es faci i la segona ens indica les dades que es reben. En les dues taules hi ha una ID que indica de quina estació meteorològica es tracta i es el paràmetre que relaciona les dues classes.

Per a poder-ho fer es va definir la columna ID de la primera taula com la clau primària de la taula, i en la segona taula, es va definir una columna amb el mateix nom com a clau forana.

D'aquesta forma es va tenir en funcionament el servidor i la base de dades. Per a afegir a la base de dades una nova estació meteorològica es va crear una vista amb l'URL *api/station/update/*, des d'on és possible afegir noves files a la base de dades mitjançant un POST en HTTP.

De la mateixa manera (amb un POST en HTTP), és com la sheild de DFRobots és capaç d'afegir les dades meteorològiques cada mitja hora a la base de dades. En aquest cas, l'enllaç URL que s'ha de fer servir és api/dades/update/.

Per a visualitzar les dades meteorològiques que s'han anat enviant, es van crear dues URLs, en una, surten totes les dades en una taula, mentre que en l'altra hi ha la possibilitat de fer un filtratge de les dades des de qualsevol columna. Aquestes URLs son *api/dades/view/* per a veure-les en forma de taula i *api/dades/filter/* per a poder fer el filtratge. El disseny d'aquestes dues vistes s'ha fet amb unes aplicacions pròpies de Django que s'anomenen *dajngo_tables2* i *django_filters* i serveixen per a la creació de taules i sistemes de filtratge de dades.

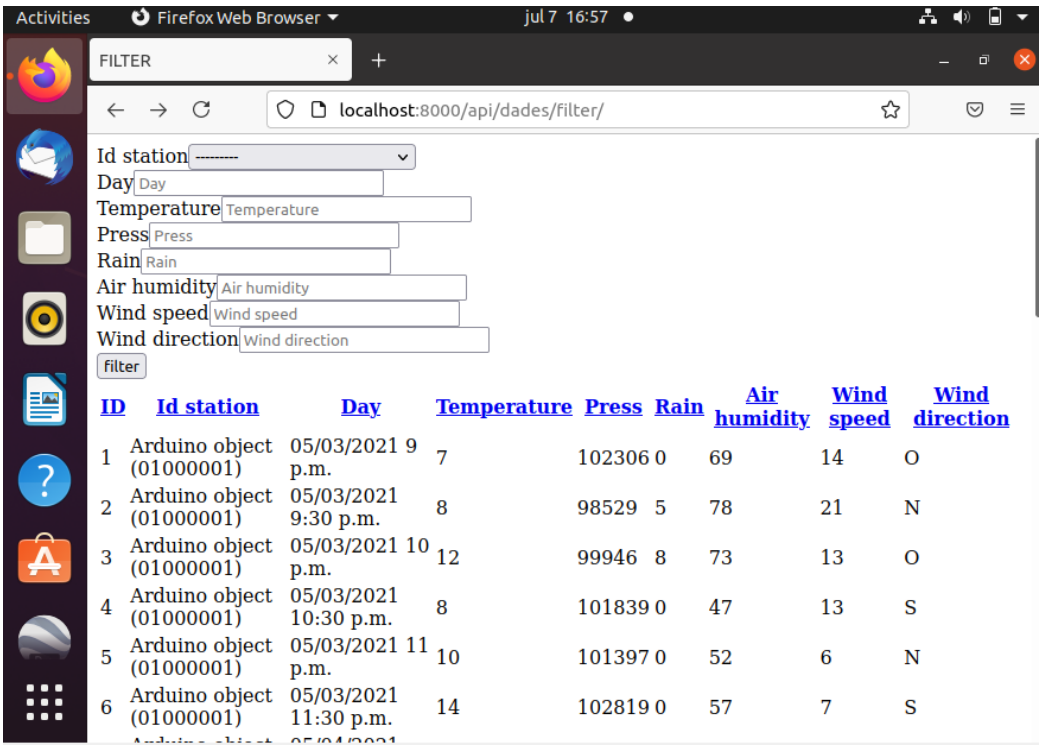


Fig. 16. Visualització en local de la URL: api/dades/view/

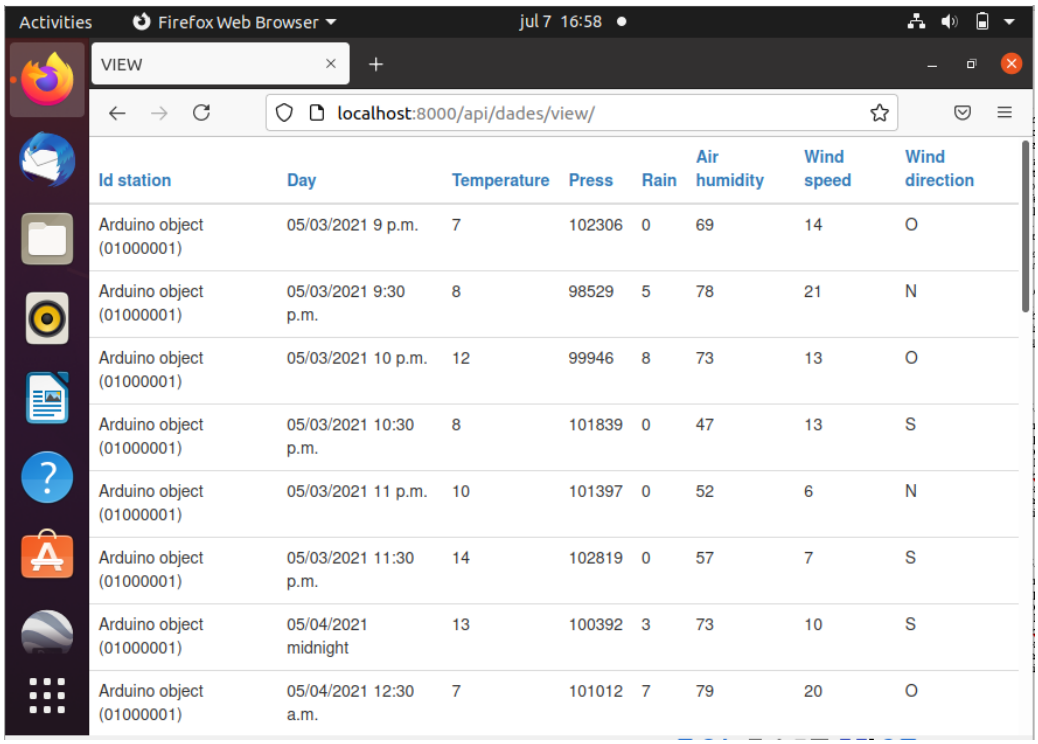


Fig. 17. Visualització en local de la URL: api/dades/filter/

El servidor ha estat creat en local. Això vol dir que no s'hi pot accedir de forma remota. Per a obrir el servidor a internet, es ha fer us d'un túnel del nostre servidor en local a alguna adreça URL oberta. Aquesta tasca es va resoldre amb ngrok; ngrok genera una URL dinàmicament, al qual apunta al servei web que s'està executant a la nostra màquina virtual.

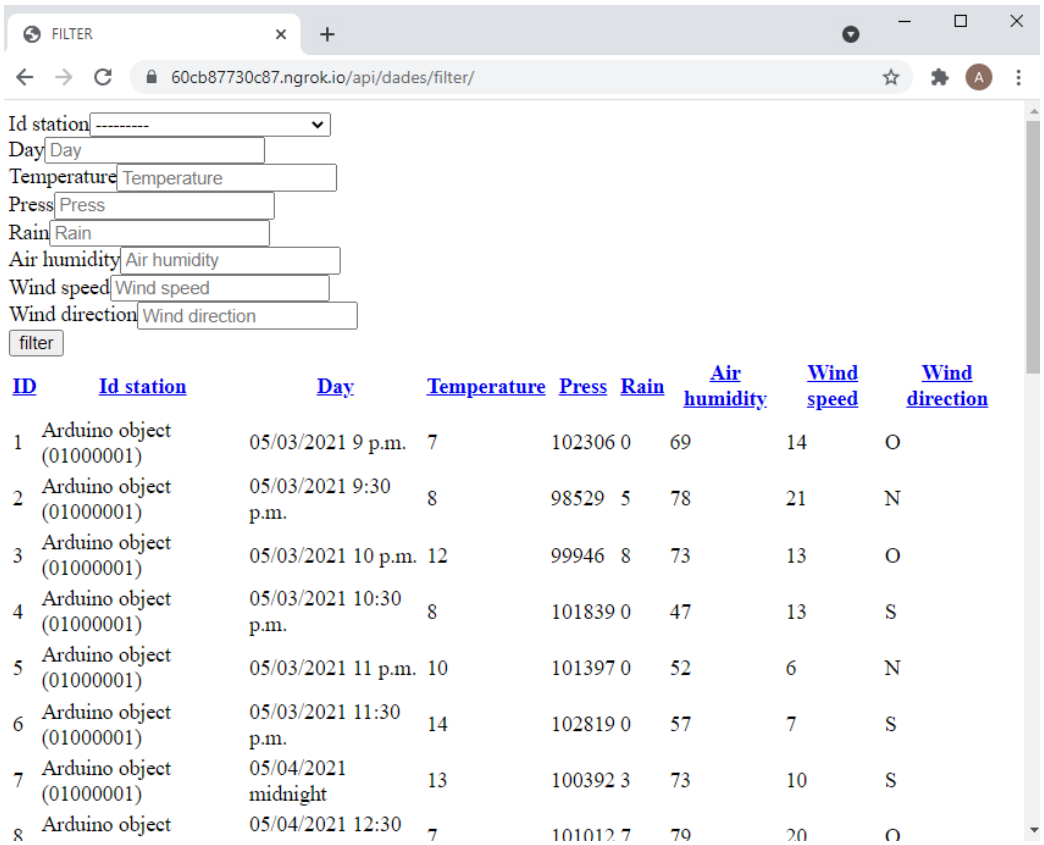
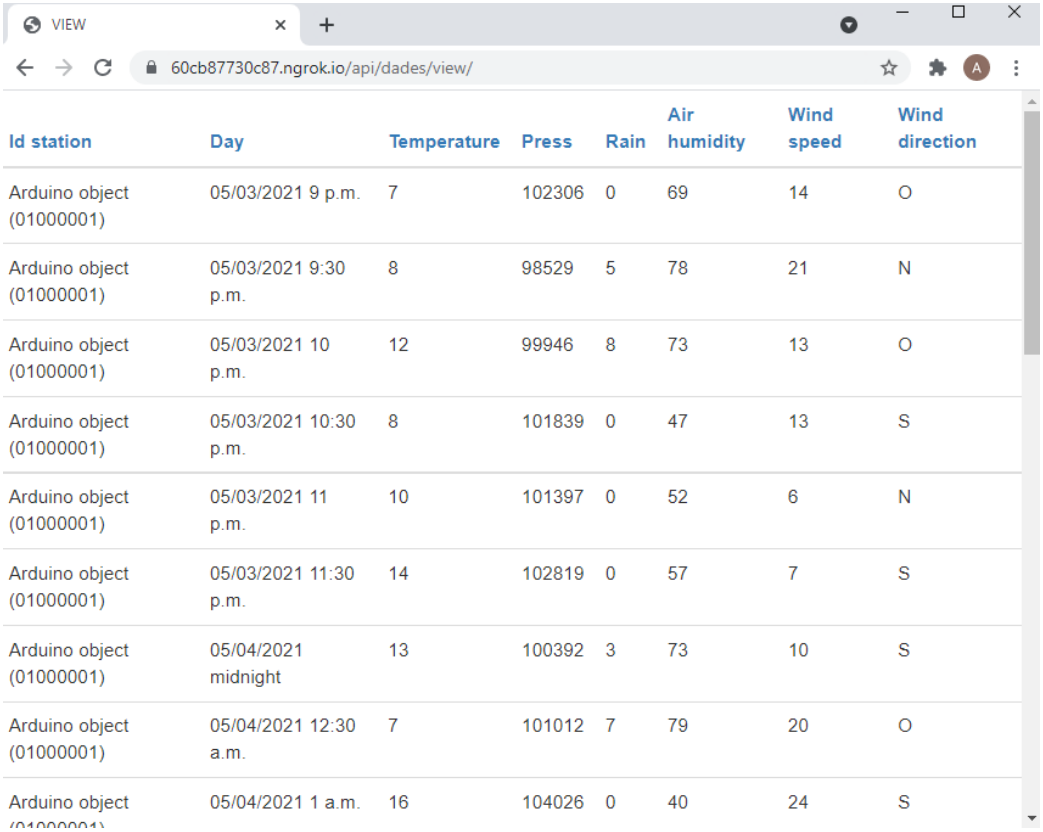


Fig. 18. Visualització en remot de la URL: api/dades/filter/ a trav´se del tunel de ngrok.io



Id station	Day	Temperature	Press	Rain	Air humidity	Wind speed	Wind direction
Arduino object (01000001)	05/03/2021 9 p.m.	7	102306	0	69	14	O
Arduino object (01000001)	05/03/2021 9:30 p.m.	8	98529	5	78	21	N
Arduino object (01000001)	05/03/2021 10 p.m.	12	99946	8	73	13	O
Arduino object (01000001)	05/03/2021 10:30 p.m.	8	101839	0	47	13	S
Arduino object (01000001)	05/03/2021 11 p.m.	10	101397	0	52	6	N
Arduino object (01000001)	05/03/2021 11:30 p.m.	14	102819	0	57	7	S
Arduino object (01000001)	05/04/2021 midnight	13	100392	3	73	10	S
Arduino object (01000001)	05/04/2021 12:30 a.m.	7	101012	7	79	20	O
Arduino object	05/04/2021 1 a.m.	16	104026	0	40	24	S

Fig. 19. Visualització en remot de la URL: api/dades/view/ a trav´se del tunel de ngrok.io

3.4.2 Client

Des d'un punt de vista d'informàtic, el client és una aplicació o sistema que fa ús d'un servei remot en un altre ordinador, anomenat servidor.

En el treball que s'ha desenvolupat, el client seran totes és estacions meteorològiques, que enviaran les dades de forma remota al servidor i s'emmagatzemaran a la base de dades; però també són clients tots els usuaris que després vulguin visualitzar les dades que s'han anat guardant.

Per al desenvolupament del projecte, la part més complicada ha estat aconseguir la comunicació entre l'Arduino i el servidor, perquè el servidor, tal com s'ha creat, només és capaç de rebre les dades que li arribin el format JSON, i el mòdul shield de DFRobots que s'ha fet servir, per defecte, envia les dades a través del POST amb format *plain-text*, per tant, el servidor retornava error de lectura perquè no era capaç de llegir la informació que li arribava. Com es va solucionar aquest problema s'explicarà més detalladament en l'Annex.

3.4.3 Arduino- mòdul DFRobots

L'Arduino desenvolupa la funció de llegir totes les dades que els sensors agafen, analitzar-les de forma que puguin ser guardades, és a dir, fer les conversions de digital a analògic en els casos que sigui necessari, i enviar-les a través de la shield de DFRobots al servidor, a través de l'enllaç *api/dades/update/*.

L'Arduino ha estat dissenyat per a aprendre a fer funcionar microcontroladors, molt enfocat a l'ensenyament i l'educació, això fa que la programació per a obtenir

les dades dels sensors és relativament senzilla, fet que fa que hi hagi moltes coses bones i algunes de dolentes. Una de les coses bones és que la conversió de digital a analògic és molt ràpida i simple: s'han de definir correctament els pins l'entrada analògica i després es llegeixen amb la funció *analogRead()* que retorna un valor depenent de l'entrada i de la resolució del convertidor que incorpori el microcontrolador.

Un dels principals problemes que tenen els Arduinos és que si es vol adormir el dispositiu per a aconseguir reduir el consum, quan es vulguin generar les interrupcions que el despertin, aquestes no podran ser periòdiques i exteriors a la vegada.

3.4.4 Usuari

L'usuari serà la persona física que es connectarà als enllaços de filtratge (*api/dades/filter/*) o visualització (*api/dades/view/*) de dades per a consultar les condicions meteorològiques que les estacions han anat llegint.

3.4.5 Procediment creació estació meteorològica

Per a poder crear l'estació meteorològica, primer de tot es va haver de valorar quins sensors es farien servir per a l'estació i, per tant, quines magnituds es mesurarien. El mòdul shield incorpora el sensor de monitoratge ambiental BME, el qual mesura la temperatura, humitat relativa i la pressió atmosfèrica. Per a la velocitat i direcció del vent i la pluja acumulada, des del parc científic em van facilitar l'estació meteorològica del fabricant SparkFun que els porta incorporats.

Un cop es va tenir clar els sensors, es va dissenyar el codi per a obtenir els valors de cadascun d'ells, fent provatures per a veure els resultats que se n'obtenien i així analitzar de forma correcta els valors que mesuraven.

El següent pas va ser provar la comunicació entre l'Arduino i el servidor. El mòdul de DFRobots necessita una targeta SIM per a funcionar, i ho pot fer a través de xarxa GPRS o NB-IoT. Per a funcionar en la xarxa GPRS es pot fer servir una targeta SIM de telefonia típica, com la que s'usa en els telèfons mòbils; pel que fa a la xarxa NB-IoT és necessària una targeta SIM que suporti aquesta xarxa, i les targetes SIM de telefonia típiques no serveixen. Per tant, es va optar per a fer la comunicació via GPRS per a poder-la fer amb la targeta d'un mòbil corrent.

Amb el codi per a obtenir les dades dels sensors i la forma de comunicació coneguda, es va començar a enviar les primeres dades al servidor, aquestes dades eren falses i el seu objectiu era el d'assegurar la comunicació amb el servidor i confirmar que el sistema funcionava.

Els primers missatges que s'enviaven al servidor no eren reconeguts perquè s'interpretaven com a *plain-text*, es va fer el possible des del codi, per a intentar que el servidor entengués que el que s'enviava era en format JSON, escrivint missatges de *headers* de HTTP, explicitant que el contingut del missatge tenia format JSON, però la resposta del servidor sempre era negativa.

Per a solucionar aquest problema, es va intentar modificar l'API creada en el servidor perquè acceptes el format *plain-text* quan s'hi fes un POST d'HTTP; però

això és un tema més d'informàtica i, després d'uns dies intentant-ho, es va desestimar per la dificultat que em suposava i perquè no en van notar millores. A continuació vaig pensar que el millor seria mirar com funcionava l'enviament de missatges POST a través de la llibreria *DFRobot_SIM7000.h*, que és la que s'encara de controlar el xip SIM7000E del mòdul de DFRobot. Amb una anàlisi en profunditat de la llibreria, es va veure que hi havia certes comandes AT (les comandes AT comuniquen la Shied amb el xip SIM7000E) que eren les encarregades de modificar els headers del protocol HTTP per tal de canviar el contingut del cos del missatge a JSON.

Buscant per internet, vaig trobar la documentació del xip SIM7000E. El següent pas va ser trobar en la documentació les comandes que servien per a fer els POST en HTTP, i un cop les tenia, mirar quina havia de fer servir per a modificar el contingut del missatge (*Content-Type*). Finalment, es va modificar la llibreria *DFRobot_SIM7000.cpp*: es va canviar la funció *httpConnect*, afegint les següents línies:

```
if(!check_send_cmd("AT+HTTPPARA=\"CONTENT\",\"application/json\"\r\n","OK")){
    return false;
}
```

```
459  bool  DFRobot_SIM7000::httpConnect(const char *Host)
460  {
461      httpDisconnect();
462      delay(5);
463      if(!check_send_cmd("AT+HTTPINIT\r\n","OK")){
464          return false;
465      }
466      if(!check_send_cmd("AT+HTTPPARA=\"CID\",\"1\"\r\n","OK")){
467          return false;
468      }
469      send_cmd("AT+HTTPPARA=\"URL\",\"");
470      send_cmd(Host);
471      if(!check_send_cmd("\"\r\n","OK")){
472          return false;
473      }
474      if(!check_send_cmd("AT+HTTPPARA=\"CONTENT\",\"application/json\"\r\n","OK")){
475          return false;
476      }
477      return true;
478  }
```

Fig. 20. Modificació del fitxer DFRobot_SIM7000.cpp

Afegint aquestes línies el que s'aconsegueix és que sempre que es faci una connexió via HTTP fent anar la funció *httpConnect* el missatge que s'enviarà haurà de ser JSON, perquè sempre s'hi afegirà un *header* dient que el contingut de la consulta serà un JSON. Una millor solució hauria estat la creació d'una nova funció dins de la llibreria que permetés modificar el contingut del missatge i cridar-la en el codi principal del programa just després de la funció *httpConnect*, indicant el tipus de contingut que s'enviarà. També s'hauria pogut escriure directament al codi principal la comanda AT del SIM7000E, però al principi del projecte es va experimentar amb el codi de prova que té la llibreria per a fer anar els missatges AT, i el funcionament no va ser del tot correcte.

Un altre inconvenient que es va tenir al programar l'Arduino, es va manifestar a l'hora d'adormir el dispositiu i despertar-lo en el moment oportú. Pel funcionament dels sensors, és necessari que es despertés cada vegada que el pluviòmetre

completi un cicle, és a dir, cada vegada que hagi de sumar 0,2794 mm a quantitat de pluja acumulada.

A part, també s'ha de despertar periòdicament per a enviar les dades al servidor. Per a adormir el dispositiu s'ha fet servir la llibreria *LowPower.h*, la qual fa servir el *watch dog* dels microcontroladors ARV per a despertar-los periòdicament. Aquesta llibreria també dóna l'opció de despertar l'Arduino de forma instantània a través d'una interrupció externa.

Aquesta llibreria té dos inconvenients principals pel treball que s'ha dut a terme: el primer és que com a molt temps, el dispositiu es pot deixar dormint durant 8 s. Per a solucionar-ho, simplement es posa un bucle definit que cada vegada que el dispositiu es desperti, el torni a adormir fins a arribar al mateix temps que es vulgui.

El segon és que si es vol despertar l'Arduino amb una interrupció externa, es despertarà al moment, sense tenir en compte l'estona que porta dormint ni l'estona que li quedava (dintre els 8 segons màxims). Això és més difícil de resoldre, tot així, es va optar per a fer un comptatge de les interrupcions que hi ha hagut cada 30 minuts, d'aquesta manera, el bucle definit es podrà allargar una mica més, per tal d'evitar que el temps d'enviament sigui molt baix.

Una altra solució que es va tenir en compte va ser la de fer servir la funció *millis()* d'Arduino. Aquesta funció retorna el temps amb milisegons que fa que el dispositiu està en funcionament, però fa servir interrupcions per a fer el càlcul, per tant, és incompatible amb la llibreria.

4 Experiments i resultats

Un dels problemes que limiten la mobilitat d'una estació meteorològica és la dificultat a l'hora de trobar una font d'alimentació que sigui suficient i que pugui durar, almenys, unes setmanes. Altres inconvenients que es poden tenir estan relacionats amb el consum de dades mòbils quan es produeixen els intercanvis d'informació entre l'Arduino i el servidor.

En una primera part d'aquest capítol s'exposaran les mesures obtingudes durant el funcionament de l'estació, tenint en compte la quantitat d'interrupcions que pot tenir l'estació a causa de la pluja, així com altres aspectes. En una segona part del capítol s'aproximaran la quantitat de dades mòbils que són necessàries per a poder fer els enviaments i els mètodes que s'han utilitzat.

4.1 Consum energètic

El càlcul del consum d'energia s'ha fet mesurant la intensitat que consumeix l'estació meteorològica en tres ocasions diferents. La primera és quan està completament adormida sense interrupcions, i durant un cert temps en el qual es va despertant cada 8 segons per a tornar-se a adormir instantàniament. La segona és quan es va despertant cada 8 segons per a tornar-se a adormir instantàniament, mentre es va despertant amb interrupcions externes. Com a últim, també s'ha mesurat quin és el consum durant l'enviament de les dades.

En tots tres mesuraments, només s'ha mesurat la intensitat, ja que el voltatge és constant a 12 volts, que són els que el fabricant recomana per al correcte funcionament del mòdul DFRobots. Així es pot obtenir la potència i capacitat que hauria de tenir una bateria que pogués fer funcionar de forma autònoma l'estació.

Els valors que s'han obtingut per cada cas han estat els que es mostren en la següent taula:

Casos	Intensitat mitjana (mA)
Dormint	55.5
Dormint amb interrupcions	55.5
Despert	120

Taula 1. Consum de l'Arduino MEGA en els diferents casos

Tal com s'observa en la taula, no s'aprecia cap mena de variació en quant està dormint, sigui amb interrupcions o no. Això és atès que el temps que roman despert quan es produeixen les interrupcions és molt baix, i no és apreciable per l'aparell de mesura que s'ha utilitzat, per tant, la diferència entre un cas i l'altre es pot considerar negligible.

Un cop es desperta i comença el procés d'obtenció de dades i el seu enviament, el consum augmenta considerablement i de forma molt variable. Aquest procés dura entre quaranta-cinc segons i un minut entre engegar-se, obtenir les dades dels sensors (són tots quasi instantanis, menys les dades de l'anemòmetre, que triga uns 5 segons a obtenir-les), enviar la informació al servidor, rebre la resposta, desactivar el mòdul SIM i tornar-se a adormir.

Tenint en compte aquests resultats, l'estació meteorològica ha d'estar-se uns vint-i-nou minuts adormida, i el minut restant desperta, amb un consum de 55,5 mA durant els vint-i-nou minuts i 120 mA en el minut que en resta. Per tant, el consum mitjà cada trenta minuts serà:

$$I_{Mitjana} = \frac{29}{30} \cdot 55.5 + \frac{1}{30} \cdot 120 = 57.65mA \approx 60mA$$

El consum mitjà total s'ha aproximat a l'alça per a impedir imprevistos en la duració de la bateria, tot i l'aproximació, el consum ja era força alt, tractant-se d'un aparell amb una font d'alimentació independent.

L'elevat consum quan es troba adormit, és degut als LED's que romanen encesos, sense possibilitat d'apagar-los, també hi afecta en gran manera que el model l'Arduino que es fa servir és un MEGA, molt més gran que un UNO o un NANO, que consumeixen menys.

Si tenim en compte aquest consum i volguéssim una duració de les bateries d'almenys un mes, la capacitat de la bateria hauria de ser:

$$1mes \cdot \frac{30\ dies}{1mes} \cdot \frac{24\ h}{1\ dia} = 720h$$

$$0.06A \cdot 720h = 43.2Ah$$

Entre els models de bateries de 12 volts, se'n troben de 45 Ah i 50 Ah. Normalment aquestes bateries estan destinades a usos d'arrencament de motors, tenen un pes força elevat i poden ser corrosives. Una altra cosa a tenir en compte és que el seu preu acostuma a ser força elevat, superant els 100 €.

Per aquestes raons, s'ha optat per una bateria de 7 Ah, no arribarà ni de bon tros a una autonomia d'un mes, però són més econòmiques i no solen ser tan corrosives. Amb una bateria d'aquest tipus, la duració total serà:

$$\frac{7Ah}{0.060A} \cdot \frac{1dia}{24h} = 4.861dies$$

4.2 Consum de dades

Per a fer el càlcul de les dades consumides durant els intercanvis d'informació sense fils entre servidor i estació meteorològica s'ha fet anar l'aplicació de *Postman*, la qual ens permet fer POST's al servidor de forma remota o local. Per tal que sigui el màxim real possible, s'ha fet servir de forma remota a través del túnel creat amb *ngrok*. S'ha optat per fer servir *Postman*, ja que directament des de l'Arduino, amb les comandes AT, o des del mateix servidor, no hi ha una opció simple i còmoda per a saber quan consumeix cada enviament.

Per a fer servir el *Postman* i obtenir la mida de la consulta i la resposta, és tan fàcil com, un cop s'ha descarregat l'aplicació de *Postman*, indicar a quina adreça es vol enviar la informació, descriure el cos del missatge i en quin format va, i enviar-ho.

El programa ens dóna la mida del missatge que s'ha enviat i rebut, per tant, la quantitat de dades que es faran servir.

Després d'unes quantes proves, s'ha observat que, evidentment, la mida del missatge depèn de les dades que els sensors captin i, a continuació, s'enviïn: com més grans siguin els valors i més xifres tinguin, més gran serà el missatge a enviar i la quantitat de dades que caldran perquè es produeixi la comunicació correctament.

Una altra cosa que s'ha observat, és que el missatge es divideix en dues parts: la primera són els *headers* i la segona el *body* (capçalera i cos del missatge). Els *headers* sempre són els mateixos i no varien, mentre que el cos del missatge sí que ho fa i és el que farà variar la mida total.

S'observa que les diferències que es poden produir en el *body* són tan minses que es podrien menysprear. Una altra cosa que s'ha de tenir en compte és que els *headers* que s'envien des de l'Arduino al servidor poden no ser exactament els que s'han especificat en el *Postman*, perquè per a saber quins s'envien a través de l'Arduino, s'ha mirat la llibreria, i podria ser que alguna comanda ens hagués passat per alt. Però tenint en compte que és un flux de dades molt minso, no és una qüestió cabdal, ja que si s'aproxima a l'alça, dient que cada enviament seran 2 kB de dades, es fan 48 missatges cada dia, durant trenta dies, es necessitaran un total de 2880 kB, que no arriba a 3 MB cada mes.

Consum (Postman):

Fig. 21. 22. Enviaments a través de Postman

5 Pressupost

En aquest treball s'ha dissenyat una estació meteorològica, centrant-se en la comunicació sense fils i el control dels sensors, per tant, s'ha de tenir en compte que el resultat final es tracta d'un prototip que mostra el funcionament, però no es tracta d'un producte funcional per a usuaris finals. Per tant, en el pressupost només es tindran en compte els materials que s'han fet servir per a fer el prototip.

Maquinari	
Arduino Mega 2560	Botiga oficial (sense impostos ni aranzels)[1]: 35€ Amazon (I.V.A inclòs)[2]: 42.35€
SIM7000E Arduino NB-IoT/LTE/GPRS/GPS Expansion Shield	Botiga oficial (sense impostos nia aranzels)[3]: 39.90€ Brico Geek (I.V.A inclòs)[4]: 48.28€
Targeta SIM de prepagament	Amazon[5]: 6.95€ + 0.0605€/MB
Sensors	
Estació meteorològica (aigua acumulada, velocitat i direcció del vent)	Brico Greek(I.V.A. inclòs)[6]: 69.00€
Sensor BME	Inclòs en SIM7000E Arduino NB-IoT/LTE/GPRS /GPS Expansion Shield
Bateria	
Bateria de 7Ah recarregable	Amazon (I.V.A. inclòs)[7]: 13.50€
Total (I.V.A. inclòs durant un mes)	179.9115€

Taula 2. Resum dels costos dels materials

[1] https://store.arduino.cc/arduino-mega-2560-rev3

[2] https://www.amazon.es/Arduino-Mega-2560-R3-Microcontrolador/dp/B0046AMGW0/ref=sr_%201_8?

[3] https://www.dfrobot.com/product-1732.html

[4] https://tienda.bricogeek.com/shields-arduino/1197-shield-lte-gprs-gps-sim7000e-para-arduino.html?gclid=CjwKCAjwoNuGBhA8EiwAFxomA4hVIMMVTOaqm0m7S_p8IpNve65nUiLF6TMEkeC67Dy2mlS3oxdk3RoCyjEQAvD_BwE

[5] https://www.amazon.es/tualarmasincuotas-es-Multiformato-Autom%C3%A1tica-Opcional-Inteligentes/dp/B081VPQQQP/ref=asc_df_B081VPQQQP/?

[6] https://tienda.bricogeek.com/sensores-humedad/672-estacion-meteorologica-con-mastil.html

[7] https://www.amazon.es/Master-U-Power-UP-Bater%C3%ADa-Plomo/dp/B084GMZYS4/ref=sr_1_5?

6 Conclusions

Una vegada s'ha construït el prototip de l'estació meteorològica és el moment de fer una repassada del que ha sigut el treball. El primer que voldria comentar és la satisfacció personal d'aconseguir superar tots els obstacles que han anat sorgint durant el període que ha durat el treball, en molts moments hi ha hagut traves que han dificultat la finalització de l'estació meteorològica, però a poc a poc i amb constància, s'han anat superant.

Entrant més en detall del projecte, s'ha de comentar que en el moment que em vaig decidir a fer el projecte, ja havia treballat amb Arduino en diverses ocasions i dominava bastant bé la seva programació i les funcionalitats que té, com ara els diversos tipus d'entrades i sortides; però no estava gens familiaritzat amb la creació de servidors, bases de dades ni programació web. Tot la incertesa inicial, vaig començar amb molta il·lusió i ganes, i tot que algunes vegades o sabia com continuar, sempre vaig disposar d'ajuda, tant per internet com per part del Guillem o del tutor, que em van guiar amb bons consells quan tenia dubtes.

Parlant dels sensors, també és interessat fer uns comentaris al respecte. És evident que per la meva formació, ja havia fet servir alguns sensors, però mai haver de buscar jo el funcionament de sensors analògics, tenint en compte que el fabricant no disposava de quasi cap mena d'informació sobre com es feien anar. De totes maneres, el funcionament que tenen no és molt complicat i la gran majoria es podien fer servir directament com a entrades digitals. Treballar amb l'Arduino també m'ha facilitat molt el treball perquè ja hi estic molt familiaritzat i perquè treballa amb una interfície que fa molt amigable la programació, a part d'una gran comunitat en la xarxa.

Pel que fa al servidor, és on he hagut de treballar de valent. Mai havia fet un treball en el qual hagués d'aplicar tants coneixements d'informàtica o combinar diversos llenguatges de programació, molt desconeguts per a mi, com Python, HTML, CSS i d'altres. Per sort he contat en tot moment amb ajuda, tant per part del tutor, guiant-me quan fos necessari, com per part del Guillem, que va ser qui em va recomanar que fes servir Django i em va dir quins llibres seguir per a començar. Altra vegada, també és d'agrair la gran comunitat de programadors que hi ha a la xarxa.

Una vegada s'ha finalitzat el treball, és moment de mirar els objectius plantejats inicialment. Es fa evident que hi ha alguns objectius que no s'han arribat a complir. Un dels objectius que es van plantejar va ser que la mateixa estació fos capaç de suportar les inclemències meteorològiques. Aquest és un objectiu que, un cop començat el treball, ni m'he plantejat de fer, per falta de temps i pel volum de feina que ha tingut el treball. Tampoc s'ha pogut fer una posada en marxa de tota l'estació i servidor per a monitorar el seu funcionament en un entorn real, ja que això hagués pogut suposar que algunes parts de l'equip es fessin malbé.

Una de les parts on també voldria tenir un esperit crític en el treball fa referència a haver fet una prèvia comparació o calibració dels sensors utilitzats amb instruments de mesura de precisió. D'aquesta manera s'haurien pogut detectar

possibles errors de mesura, cosa que és molt important quan es treballa amb instruments de mesura.

Un altre objectiu que no s'ha pogut assolir ha estat que la comunicació entre l'estació meteorològica i el servidor fos a través d'una xarxa MQTT i no la convencional GPRS, que tot hi ser igualment molt lleugera, i així s'ha demostrat a l'apartat de consum de dades, no és tan ràpida. De totes maneres el treball es pot englobar dintre d'un projecte IoT (Internet of Thinks).

Una última crítica que voldria fer al treball és la pàgina web. La idea que inicialment tenia al cap, era crear una gràfica on es pogués elegir les magnituds que es volien veure en funció de les dates, i s'hi mostrés les mitjanes diàries. Finalment, això no ha pogut ser i per a visualitzar les dades s'ha fet una taula on es pot filtrar per cada valor de magnitud. Tot hi aquest comentari, s'ha de puntualitzar, i que quedi clar, que estic molt content i satisfet amb el servidor web que he creat i amb la base de dades; sobretot tenint en compte que quin punt es partia, jo mai havia programat un servidor i no sabia quasi res del funcionament, i a on s'ha acabat.

Pel que fa al seguiment, la panificació i la metodologia seguida, s'ha de comentar que s'ha seguit bastant al peu de la lletra amb el que s'havia planificat inicialment, començant amb una breu recerca sobre els materials que s'havien d'utilitzar i els fonaments dels conceptes meteorològics. A continuació, es va treballar amb el servidor i la base de dades; aquí és on més estona vaig estar, entre mirar quina estructura de web fer servir i com programar-la. En paral·lel amb això, es va anar recollint els sensors per a conèixer el funcionament de cada un, fent proves sobre el funcionament juntament amb l'Arduino. Un cop el servidor va estar en marxa, es va començar amb la comunicació entre servidor i Arduino, trobant diversos problemes, però es van poder solucionar amb bastant agilitat. Finalment, es va fer el muntatge del prototip i se'n va comprovar el correcte funcionament, donant per finalitzat el treball.

És important comentar que normalment planifiques el treball amb temps i, sol passar, principalment degut als contratemps que van sorgint, que s'ha de replanificar tota la planificació que s'havia fet inicialment. Això ha succeït les diverses ocasions, ja que uns quants obstacles han anat sorgint.

Per a finalitzar aquest apartat de conclusions, voldria agrair tot el suport rebut per amics, familiars i companys d'universitat, els quals han estat en tot moment per a ajudar-me en qualsevol dubte que ha anat sorgint. Una menció especial es mereixen el Guillem Felis, company als laboratoris Liquid Galaxy i estudiant d'informàtica, que m'ha ajudat molt en la creació del servidor; l'Andreu Ibàñez, responsable dels laboratoris Liquid Galaxy, ajudant-me en la visió global del treball i amb els materials; i el tutor del TFG, el Tomàs Pallejà, qui sempre hi ha estat quan he tingut qualsevol dubte, de qualsevol mena.

7 Treball futur

Com a línia de treball futura, i en especial, s'ha de construir una estructura que permeti que l'estació pugui suportar les inclemències meteorològiques, d'aquesta manera, es tindrà una estació mínimament funcional. Aquest es un dels objectius inicials que no s'han pogut completar.

Un altre punt a millorar, relacionat amb el seu funcionament en la intempèrie, seria unes bateries suficients, en el treball només s'han fet els càlculs, però s'hauria de comprovar si realment són suficients; aquestes bateries, també es podrien millorar amb un equip de plaques solars fotovoltaiques, millorant molt més l'autonomia.

Relacionat amb les bateries, també estaria molt bé mirar altres maneres de fer adormir el microcontrolador, o buscar microcontroladors que tinguessin un consum més baix, per tal d'encara poder estalviar més energia, tenir un funcionament més llarg i ser un sistema molt més eficient.

La part d'usuari és un altre aspecte que es podria millorar. Django disposa d'un sistema molt potent i molt fàcil d'implementar per a la creació de comptes d'usuaris, on es poden afegir moltes funcionalitats. Retocar aquest aspecte faria que el projecte fos cent per cent funcional per a usuaris finals.

En la mateixa línia, estaria molt bé poder millorar la interfície gràfica de cara a la persona física que ho ha de consultar, afegint gràfiques on es deixi triar les magnituds i les dates i on es puguin veure les mitjanes diàries, mensuals i/o anuals.

Com a millora final, es podrien afegir alertes via SMS al mòbil de l'usuari en cas de condició meteorològica extrema, com ara temperatures negatives i baixa humitat relativa, o forts vents (per a enviar i rebre missatges, s'han de fer servir les comandes AT, no hi ha cap funció ja feta a la llibreria).

8 Referencies

Documentació Django: https://docs.djangoproject.com/en/3.2/

Documentació Django REST Framework:
https://www.django-rest-framework.org/

Llibre *Django for Begginers, William S. Vincent (Molt important a l´hora de programar el servidor):*
https://es1lib.org/book/3594011/891d30?dsource=recommend

Llibre *Django for Proffesionals, William S. Vincent:*
https://es1lib.org/book/5288427/a89fa8?dsource=recommend

Llibre *Django for APIs: Build Web APIs with Python & Django, William S. Vincent:* https://es1lib.org/book/5288428/ac2f5b

Aplicació externa de Django, Django-filters:
https://pypi.org/project/django-filter/

Aplicació externa de Django, Django-tables2:
https://django-tables2.readthedocs.io/en/latest/

ArduinoJson: https://arduinojson.org/

Documentació Arduino: https://www.arduino.cc/reference/en/

Documentació SIM7000E DFRobots:
https://www.dfrobot.com/product-1732.html
https://wiki.dfrobot.com/SIM7000_Arduino_NB-IoT_LTE_GPRS_Expansion_Shield_SKU__DFR0505_DFR0572?gclid=Cj0KCQjw24qHBhCnARIsAPbdtlKSjM0F1rY9rbzA5WTBmM5RDDHh2JT_6gcHzONzYxMeN36QRAiOPHQaAhElEALw_wcB

Manual_V1.01 SIM7000 Series_AT Command Manual_V1.01:
https://simcom.ee/documents/SIM7000x/SIM7000%20Series_AT%20Command%20Manual_V1.04.pdf

LowPower: https://www.prometec.net/el-modo-sleep-en-arduino/

Llibreria DFRobot_BME280: https://github.com/DFRobot/DFRobot_BME280

Llibreria DFRobot_SIM7000: https://github.com/DFRobot/DFRobot_SIM7000

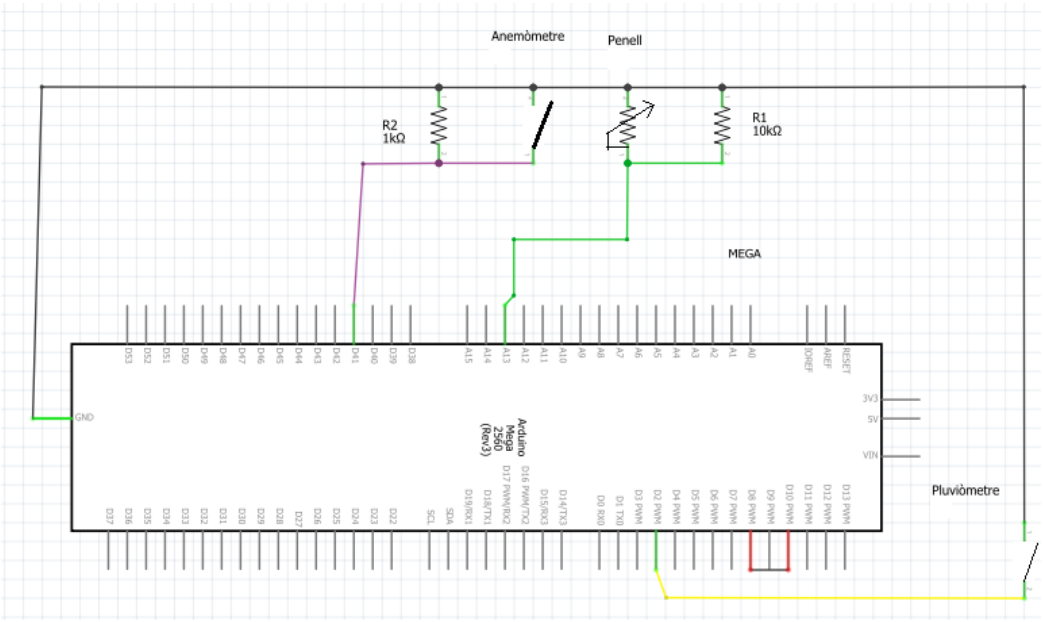
Llibreria DFRobot_SIM: https://github.com/DFRobot/DFRobot_SIM

Documentació estació meteorològica:
https://www.sparkfun.com/datasheets/Sensors/Weather/Weather%20Sensor%20Assembly..pdf

9 Annex

9.1 Esquema elèctric

En la següent figura es mostra l'esquema elèctric de les connexions. No s'hi mostra cap connexió entre la Shield, ja que simplement s'ha de posar una sobre l'altra encaixant les potes. Tot i això, s'han de puntejar el pins D8 i D10 i la comunicació Arduino- Shield flueix a través dels pins D8, com a TX, i D7, com a RX, els dos des del punt de vista de la Shield.



Esquema elèctric connexions amb els sensors de l'estació meteorològica

L'anemòmetre i el pluviòmetre es comporten com interruptors, per això es consideren directament sensors digitals en entrades digitals. En l'esquema podem veure que en el cas de l'anemòmetre, s'ha implementat una resistència *pull down* perquè l'entrada es trobi a nivell baix cada vegada que el sensor (interruptor es troba obert), en el cas del pluviòmetre, hi ha una resistència *pull up* implementada internament en l'Arduino des de codi.

El penell, en canvi, genera un divisió de tensió, ja que depenent de la direcció on apunti té una resistència o una altra, actuant com una resistència variable o potenciòmetre.

9.2 Codi Arduino

En el següent requadre es mostra el codi que s'ha fet servir en l'Arduino. Prèviament, s'han d'haver instal·lat les llibreries que estan incloses. Totes les llibreries menys *Wire.h* i *ArduinoJson.h* s'han de descarregar des de la web, no es pot fer a través de l'administrador de biblioteques del ID Arduino.

A part de les que s'inclouen, també és necessari instal·lar el paquet de llibreries *DFRobot_SIM.h* i modificar, tal com s'ha explicat anteriorment, la llibreria *DFRobot_SIM7000.h.*

```
/*
  * File  : DFRobot_SIM7000_HTTP.ino
  * Power : SIM7000 needs 7-12V DC power supply
  * Brief : This example verify HTTP connection over a HTTP request
  *          With initialization completed, we connect to server POST
data and GET data
  *         Thus we finished the HTTP POST and GET verification
  * Note  : If you use NB-IOT please confirm that the IOT is in the
whitelist of your NB card
  *         If you use Mega please connect PIN8 PIN10 and set PIN_RX =
10
  */

#include <Wire.h>
#include <DFRobot_SIM7000.h>
#include "DFRobot_BME280.h"
#include "LowPower.h"
#include <ArduinoJson.h>


#define PIN_TX     7
#define PIN_RX     10

#define ID_Arduino  "10000000"

//This URL is use for post data to tlink (s'ha de canvair, depèn del
túnel)
#define POSTURL "b5afed164d5d.ngrok.io/api/dades/update/"

// Vbles control parades
int i = 0;
int valor_max = 4;
int stops = 0;

SoftwareSerial     mySerial(PIN_RX,PIN_TX);
DFRobot_SIM7000    sim7000;

typedef DFRobot_BME280_IIC    BME;
BME   bme(&Wire, 0x76);
#define SEA_LEVEL_PRESSURE    1015.0f

void (*resetFunc)(void)=0;

// PINs de connexió Arduino MEGA
const byte vent = 41;
const byte aigua = 2;
int veleta =  A13;

// Vbles pluviometre  G i R (Un al negatiu)
const float incrementAigua = 0.2794;
float quantitatAigua = 0;

// Vbles anemòmetre   N i G
unsigned long temps;
float incrementT = 5000;
float cont = 0;
float contAnt = 0;
float mostres = 0;
float rFactor = 2.4;
float velocitat = 0;
// N a positiu, G pota positiva resistencia
// connexió amb sèrie, cable a arduino per allà

// Vbles penell     V i R
float volt = 235;
String direccio = "N";
// V a positiu, R pota positiva resistencia
// connexió amb sèrie, cable a arduino per allà
```

```
void SIM(){
    bme.reset();
    Serial.println("bme read data test");
    while(bme.begin() != BME::eStatusOK) {
      Serial.println("bme begin faild");
      printLastOperateStatus(bme.lastOperateStatus);
      delay(2000);
    }
    Serial.println("bme begin success");
  delay(100);

  float   temp = bme.getTemperature();
  uint32_t    press = bme.getPressure();
  float   humi = bme.getHumidity();

  Serial.println();
  Serial.println("======== start print ========");
  Serial.print("temperature (unit Celsius): "); Serial.println(temp);
  Serial.print("pressure (unit pa):         "); Serial.println(press);
  Serial.print("humidity (unit percent):    "); Serial.println(humi);
  Serial.println("========  end print  ========");

  delay(1000);

    int signalStrength,dataNum;
    sim7000.begin(mySerial);
    Serial.println("Turn ON SIM7000......");
    if(sim7000.turnON()){                                    //Turn ON
SIM7000
        Serial.println("Turn ON !");
    }

    Serial.println("Set baud rate......");
    while(1){
        if(sim7000.setBaudRate(19200)){                          //Set
SIM7000 baud rate from 115200 to 19200 reduce the baud rate to avoid
distortion
            Serial.println("Set baud rate:19200");
            break;
        }else{
            Serial.println("Faile to set baud rate");
            delay(1000);
            resetFunc();
        }
    }

    Serial.println("Check SIM card......");
    if(sim7000.checkSIMStatus()){                               //Check
SIM card
        Serial.println("SIM card READY");
    }else{
        Serial.println("SIM card ERROR, Check if you have insert SIM
card and restart SIM7000");
        resetFunc();
    }

    Serial.println("Set net mode......");
    while(1){
        if(sim7000.setNetMode(GPRS)){                        //Set net
mod GPRS
            Serial.println("Set GPRS mode");
            break;
        }else{
            Serial.println("Fail to set mode");
            delay(1000);
            resetFunc();
```

```
        }
    }

    Serial.println("Get signal quality......");
    signalStrength=sim7000.checkSignalQuality();                //Check
signal quality from (0-30)
    Serial.print("signalStrength =");
    Serial.println(signalStrength);
    delay(500);

    Serial.println("Attaching service......");
    while(1){
        if(sim7000.attacthService()){                          //Open the
connection
            Serial.println("Attach service");
            break;
        }else{
            Serial.println("Fail to Attach service");
            delay(1000);
            resetFunc();
        }
    }

    Serial.println("Init http......");
    while(1){
        if(sim7000.httpInit(GPRS)){                              //Init
http service
            Serial.println("HTTP init !");
            break;
        }else{
            Serial.println("Fail to init http");
            resetFunc();
        }
    }

    velocitatVent();
    direccioVent();

    Serial.print("POST to ");
    //Serial.println(POSTURL);
    String httpbuff;
    // Creació del document JSON amb la llibreria ArduinoJson
    DynamicJsonDocument doc(1024);
    doc["id_station"] = ID_Arduino;
    doc["temperature"] = String(temp);
    doc["press"] = String(press);
    doc["rain"] = String(quantitatAigua);
    doc["air_humidity"] = String(humi);
    doc["wind_speed"] = String(velocitat);
    doc["wind_direction"] = direccio;

    serializeJsonPretty(doc,httpbuff);      // Obtenció del JSON en
l'string output
    Serial.print(httpbuff);

    while(1){
        if(sim7000.httpPost(POSTURL,httpbuff)){                  //HTTP
POST
            Serial.println("Post successed");
            break;
        }else{
            Serial.println("Fail to post");
        }
    }

    Serial.println("Disconnect");
    sim7000.httpDisconnect();                                 //Disconnect
    Serial.println("Close net work");
```

```
    sim7000.closeNetwork();                                  //Close
net work
    Serial.println("Turn off SIM7000");
    sim7000.turnOFF();                                      //Turn OFF
SIM7000
}

void printLastOperateStatus(BME::eStatus_t eStatus)
{
  switch(eStatus) {
  case BME::eStatusOK:    Serial.println("everything ok"); break;
  case BME::eStatusErr:   Serial.println("unknow error"); break;
  case BME::eStatusErrDeviceNotDetected:    Serial.println("device not
detected"); break;
  case BME::eStatusErrParameter:    Serial.println("parameter error");
break;
  default: Serial.println("unknow status"); break;
  }
}

void velocitatVent(){
  mostres = 0;
  temps = millis();
  while (millis()<= temps+incrementT){
    cont = digitalRead(vent);
    if (cont == LOW && contAnt == HIGH){
      mostres = mostres + 1;
    }
    contAnt = cont;
  }
  velocitat = mostres*1000*rFactor/incrementT;
}

void direccioVent(){ // en Nord el senyala el pluviomentre
  volt = analogRead(veleta);
  if (volt > 230 && volt < 245){
    direccio = "N";
  }
  else if(volt > 725 && volt < 745){
    direccio = "S";
  }
  else if(volt > 920 && volt < 940){
    direccio = "E";
  }
  else if(volt > 65 && volt < 85){
    direccio = "W";
  }
  else if(volt > 500 && volt < 560){
    direccio = "NE";
  }
  else if(volt > 120 && volt < 145){
    direccio = "NW";
  }
  else if(volt > 790 && volt < 850){
    direccio = "SE";
  }
  else if(volt > 360 && volt < 400){
    direccio = "SW";
  }
}


void pluja(){
  detachInterrupt(digitalPinToInterrupt(aigua));
  quantitatAigua = quantitatAigua + incrementAigua;
  attachInterrupt(digitalPinToInterrupt(aigua), pluja, LOW);
}
```

```
void setup() {
  Serial.begin(9600);
  while(!Serial){}

  pinMode(vent,INPUT);
  pinMode(aigua, INPUT_PULLUP);
  pinMode(veleta,INPUT);

  attachInterrupt(digitalPinToInterrupt(aigua), pluja, LOW);
  SIM();
}

void loop() {
  LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
  if(i > valor_max){
    SIM();
    i = 0;
    quantitatAigua = 0;
  }
  else{
    i++;
  }
}
```

9.3 Codi servidor

El servidor fa servir l'estructura web de Django, la qual està basada en Python. L'organització interna està formada per diferents directoris. En la següent imatge es mostra l'organització en forma d'arbre. Seguidament, s'adjuntaran els codis dels fitxers que s'han de modificar, la resta són generats automàticament en el procés de creació del servidor.

```
albert@albert-VirtualBox:~/Desktop/Prova3$ tree
.
├── data
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   │   ├── 0001_initial.py
│   │   └── __init__.py
│   ├── models.py
│   ├── serializers.py
│   ├── tables.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── db.sqlite3
├── manage.py
├── Pipfile
├── Pipfile.lock
├── Procfile
├── Prova3
│   ├── asgi.py
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── README.md
└── Templates
    ├── data.html
    └── filter.html

4 directories, 24 files
```

Estructura en forma d'arbre del projecte en Django

L'única cosa que s'ha de fer de més a més és la creació de la'plicació Django, on hi haurà la base de dades i des d'on es mostraran les vistes i enllaços que es faran sevir. En aquest cas, li he posat data.

Per la creació d'una aplicació amb Django, es pot mirar directament la documentació de Django o seguir el llibre *Django for Begginers* (és el procediment que vaig seguir jo), ambdós es troben en la bibliografia.

9.3.1 Settings.py

El fitxer Settings.py és on mirarà el servidor cada vegada que es posi en marxa. És important afegir, a *ALLOWED_HOSTS*, la nova adreça que proporciona *ngrok* cada vegada, per tal de poder accedir al servidor remotament.

```
import os
#import CustomParser

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
#                                                                    See
https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY                    =                    'django-insecure-
cgyb#pt9=hdn_x6*5k8mz!wt4lxtcocpm%nwtmtp-39d&n^y2&'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS   =   ["localhost",   "127.0.0.1",   "0.0.0.0",   "[::1]",
"f9adb489a0ae.ngrok.io"]


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # 3rd part
    'rest_framework',
    'crispy_forms',
    'bootstrap5',
    'django_tables2',
    'django_filters',
    'bootstrap3',

    # Local
    #'dades.apps.DadesConfig',
    'data.apps.DataConfig',
    #'Pages.apps.PagesConfig',
]

REST_FRAMEWORK = {
            'DEFAULT_PARSER_CLASSES': [
                        'rest_framework.parsers.JSONParser',
```

```
                            #'CustomParser.PlainTextParser',
            ],
}

CRISPY_TEMPLATE_PACK = 'bootstrap5'

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'Prova3.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'Templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'Prova3.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
#   https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-
validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValida
tor',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
```

```
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = False

USE_TZ = False


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'

# Default primary key field type
#    https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-
field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

DATETIME_FORMAT="%Y-%m-%d %H:%M:%S"

APPEND_SLASH=False
```

9.3.2 Urls.py

Aquest fitxer urls.py es el principal, on es gestionen tots els enllaços del servidor.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    #path('', include('Pages.urls')),
    path('admin/', admin.site.urls),
    path('api/', include('data.urls')),
]
```

9.3.3 Models.py

En el fitxer models.py, s'hi defineixen les taules de la base de dades, que seran classes de Python. Com es pot veure, es van crear dos classes, una fa referència a cada estació meteorològica, anomenada Arduino, i l'altra son les dades que es van enviant.

En la classe *Data* hi ha un valor que te l'argument *ForeignKey*, que fa referència a la classe Arduino. D'aquesta manera, queden les dues classes relacionades, de tal manera que cada enviament de dades ha de relacionar-se amb alguna estació meteorològica.

```
from django.db import models
from datetime import datetime

# Create your models here.


class Arduino(models.Model):
    id_station = models.CharField(max_length=8, primary_key=True)
    model_arduino = models.CharField(max_length=50)
    lat = models.CharField(max_length=50)
    lon = models.CharField(max_length=50)
    country = models.CharField(max_length=50)
    region = models.CharField(max_length=50)
    owner = models.CharField(max_length=50)

class Data(models.Model):
    id_station = models.ForeignKey('Arduino',
on_delete=models.CASCADE)
    day = models.DateTimeField(default=datetime.now)
    temperature = models.CharField(max_length=50)
    press = models.CharField(max_length=50)
    rain = models.CharField(max_length=50)
    air_humidity = models.CharField(max_length=50)
    wind_speed = models.CharField(max_length=50)
    wind_direction = models.CharField(max_length=50)

    def __str__(self):
        return self.dia
```

9.3.4 Serializers.py

El fitxer serializers.py permet convertir les dades de la base de dades a un altre format fàcilment renderitzable per el llenguatge Python, com podrien ser JSON o XML

```
from rest_framework import serializers
from .models import Data, Arduino


class ArduinoSerializer(serializers.ModelSerializer):
    class Meta:
        fields = ('id_station', 'model_arduino', 'lat', 'lon',
'country', 'region', 'owner')
        model = Arduino


class DataSerializer(serializers.ModelSerializer):
    class Meta:
        fields = ('id_station', 'day', 'temperature', 'press', 'rain',
'air_humidity', 'wind_speed', 'wind_direction')
        model = Data
```

9.3.5 Views.py

En aquest fitxer es creen les vistes que es volen mostrar a cada url.

```
from django.shortcuts import render
from rest_framework import generics
from rest_framework.views import APIView

from django_tables2 import SingleTableView
from django_filters.views import FilterView
from django_tables2.views import SingleTableMixin

from django.views.generic import ListView
#from graphos.sources.modle import ModelDataSource

from .models import Data, Arduino
from .tables import DataTable
from .serializers import DataSerializer, ArduinoSerializer

# Create your views here.

class DataList(generics.ListCreateAPIView):
    queryset = Data.objects.all()
    serializer_class = DataSerializer

class DataTable(SingleTableView):
             model = Data
             table_class = DataTable
             template_name = 'data.html'

class FileredDataView(SingleTableMixin, FilterView):
             table_calss = DataTable
             model = Data
             template_name = 'filter.html'

class ArduinoList(generics.ListCreateAPIView):
    queryset = Arduino.objects.all()
    serializer_class = ArduinoSerializer
```

9.3.6 Urls.py

Aquest fiter urls.py es el de l'aplicació *data*, que fa la funció d'API, encarregada de rebre i mostrar totes les dades. Dona a cada vista una url.

```
from django.urls import path

from .views import DataList, DataTable, FileredDataView, ArduinoList

urlpatterns = [
    path('dades/update/', DataList.as_view()),
    path('dades/view/', DataTable.as_view()),
    path('dades/filter/', FileredDataView.as_view()),
    path('station/update/', ArduinoList.as_view()),
]
```

9.3.7 Tables.py

El fitxer tables.py s'ha de crear per a poder mostrar les dades en una taula i en el mode de filtratge. Es fa servir dues aplicacions de tercers, especificades en el fitxer Settings.py: *django_tables2* i *django_filters*

```
import django_tables2 as tables
from .models import Data

class DataTable(tables.Table):
            class Meta:
                        model = Data
                        template_name =
"django_tables2/bootstrap.html"
                        fields = ("id_station", "day",
"temperature", "press", "rain", "air_humidity", "wind_speed" ,
"wind_direction")
```

9.3.8 Filter.html

Aquest fitxer .html es troba dins del directori *Templates*, que es on s'hi col·loquen els fitxers .html que s'han de fer servir i així està especificat en el fitxer *Settings.py*. La funció que te es la de mostrar la pagina de filtratge de dades.

```
{% load render_table from django_tables2 %}
{% load bootstrap3 %}

<!doctype html>
<html>
    <head>
        <title>FILTER</title>
    </head>
</html>

{% if filter %}
    <form action="" method="get" class="form form-inline">
        {% bootstrap_form filter.form layout='inline' %}
        {% bootstrap_button 'filter' %}
    </form>
{% endif %}
{% render_table table 'django_tables2/bootstrap.html' %}
```

9.3.9 Data.html

Aquest .html es troba situat en el mateix directori que l'anterior per la mateixa raó, i la seva funció es la de mostrar la taula amb totes les dades.

```
{# tutorial/templates/tutorial/people.html #}
{% load render_table from django_tables2 %}
<!doctype html>
<html>
    <head>
        <title>VIEW</title>
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.mi
n.css" />
    </head>
    <body>
        {% render_table table %}
    </body>
</html>
```